Core Data не проектировался с целью заменить все возможные варианты хранения данных, которые могут лучше подойти при решении определенных задач (например, хранение неструктурированной информации в файлах или хранение пар значений вида ключ-значение в специализированных СУБД), однако он тесно интегрирован с другими фреймворками от Apple, например с Cocoa Touch. Core Data скрывает большинство деталей по работе с хранилищем данных, позволяя разработчику сконцентрироваться на описании бизнес-логики приложения.

А.Е. Руденко (УО «ГГУ им. Ф. Скорины», Гомель)
Науч. рук. **П.Л. Чечет,** канд. техн. наук

ИСПОЛЬЗОВАНИЕ ШАБЛОНА ПРОЕКТИРОВАНИЯ «ПУЛЛ ОБЪЕКТОВ» В ИГРОВОМ ПРИЛОЖЕНИИ

Объектный пул (англ. object pool) – порождающий шаблон проектирования, набор инициализированных и готовых к использованию объектов. Когда системе требуется объект, он не создаётся, а берётся из пула. Когда объект больше не нужен, он не уничтожается, а возвращается в пул.

Шаблон проектирования используется для улучшения производительности и эффективности использования памяти за счет повторного использования объектов из фиксированного пула, вместо их индивидуального выделения и освобождения.

Программирование для игровых консолей или мобильных телефонов больше похоже на программирование для встроенных систем, чем на программирование для PC. Как и в программировании для встроенных систем, консольные или мобильные игры должны работать очень долгое время без падений и утечек памяти, при том что эффективные менеджеры памяти встречаются не так уж и часто. В такой рабочей среде фрагментация памяти смертельно опасна.

Даже если фрагментация встречается нечасто, она может постепенно привести кучу (heap) в состояние бесполезных пузырей из дырок и щелей, полностью лишив игру возможности с ней работать.

Большинство консольных платформодержателей требуют, чтобы игры проходили «тест на протечку» (soak test), когда игра оставляется работающей на несколько дней в демо-режиме. Если игра падает – ей не разрешают выйти. Тест на протечку иногда проваливается и из-за какой-нибудь редкой ошибки, но чаще всего игру обрушивает утечка памяти, вызванная фрагментацией.

Пул объектов обладает следующими преимуществами: с точки зрения менеджера памяти он представляет собой единую непрерывную область в памяти, освобождаемую только после завершения игры; с точки зрения программиста (пользователя пула), он предоставляет простую возможность создавать и удалять объекты.

Этот шаблон широко используется в играх не только для очевидных вещей типа игровых сущностей и визуальных эффектов, но и для менее заметных структур данных типа проигрываемых звуков.

Пул объектов можно использовать, когда программе нужно часто создавать и удалять однотипные объекты, либо, если выделение объектов из кучи работает медленно или может привести к фрагментации памяти, или, например, когда каждый объект инкапсулирует ресурс типа базы данных или сетевого соединения, который сложно получать и можно использовать повторно.

Размер пула необходимо настраивать соразмерно с нуждами игры. При настройке обычно проще всего понять когда пул недостаточного размера. В тоже время, необходимо следить и за тем чтобы пул не был слишком большим. Если уменьшить пул, освободившуюся память можно использовать для чего-либо более полезного.

Одним из ограничений пула является то, что в каждый момент времени может быть активно только определенное (заранее известное) количество объектов. Разделение памяти на отдельные пулы для различных типов объектов означает с одной стороны то что последовательность взрывов не заставит, к примеру, систему частиц использовать всю доступную память, не позволив программисту создать на игровой сцене что-либо более полезное, вроде нового противника.

Кроме того, программист должен быть готов к ситуации, когда выделить объект из пула не удается, потому что все объекты заняты. Есть несколько стратегий обработки такой ситуации.

Прямое вмешательство. Это самое очевидное «исправление»: будем настраивать размер пула таким образом, чтобы он никогда не переполнялся независимо от действий пользователя. Для пулов с важными объектами, такими как противники или геймплейные предметы, это хороший выход. Не может быть «правильной» обработка недостатка свободных слотов для создания большого босса, когда игрок дошел до конца уровня. Недостатком этого метода является то, что программисту придется держать занятыми большие объемы памяти ради каких-то редких крайних случаев. Поэтому фиксированный размер пула не может считаться лучшим решением для всех состояний игры. Например, некоторые игровые уровни могут требовать расширенных пулов для визуальных эффектов, а другие уровни – для звуков.

В таких случаях лучше иметь пулы объектов, настраиваемые отдельно для каждого из сценариев.

Игнорирование создания объекта. Тип решения, который имеет смысл для некритичных частей игры, например, при работе с системами частиц. Пользователь скорее всего не обратит внимание если следующий взрыв будет немного менее впечатляющим, чем уже отображаемые на экране.

Принудительное уничтожение существующего объекта. Для примера можно использовать полностью заполненый пул проигрываемых в данный момент в игре звуков. При этом планируется, что игнорирование создания нового звукового эффета будет негативно воспринято пользователем, который уже привык слышать подобные звуки на определенные действия (например, на действие перезарядки оружия). В данном случае есть смысл найти самый тихий звук из тех, что уже играются и заменить его новым звуком. Новый звук заглушит слышимый обрыв предыдущего звука. В целом, если исчезновение существующего объекта будет менее заметным чем непоявление нового – это вполне хорошее решение.

Увеличение размера пула. В случае, если менеджер памяти позволяет программисту распоряжаться памятью более гибко, он можете увеличивать размер пула во время выполнения или создавать дополнительные пулы переполнения. Однако динамические пулы страдают от всех недостатков обычного управления динамическими структурами данных и могут быть неприменимы в случае с ограничениями на производительность игровой платформы.

Большинство менеджеров памяти обладают отладочными функциями, которые очищают только что выделенную или освобожденную память определенными значениями, к примеру **0xdeadbeef**. Это помогает обнаруживать болезненные ошибки, вызванные использованием неинициализированных значений или обращением к уже освобожденной менеджером памяти.

Так как наш пул объектов не заходит в деле управления памяти дальше повторного использования объектов, он не имеет подобной «страховочной сетки». Еще хуже то, что память, используемая для нового объекта хранила раньше объект того же самого типа. Это делает весьма возможной ситуацию, когда программист забудет инициализировать что-то внутри нового созданного объекта, а память, где он будет размещаться, уже будет содержать почти корректные данные, оставшиеся от прошлого объекта. Необходимо с особой тщательностью следить за тем, чтобы код инициализации нового объекта в пуле выполнял инициализацию объекта полностью.

Пулы объектов реже всего используются в системах со сборщиками мусора, потому что в таком случае менеджер памяти сам занимается проблемой фрагментаци. Но пулы все равно полезны тем, что помогают программисту избегать выделения и освобождения памяти. Особенно актуально подобное выделение памяти на мобильных устройствах с медленным процессором и простым сборщиком мусора.

В этом случае следует опасаться потенциальных конфликтов. Так как пул на самом деле не освобождает объекты когда они больше не используются, они остаются в памяти. Если они содержат ссылки на другие объекты, они тем самым не дадут сборщику утилизировать и эти объекты тоже. Чтобы этого избежать, нужно очищать все ссылки на другие объекты, когда объект из пула нам больше ненужен.

Е.А. Рушнов (БГУ, Минск)
Науч. рук. **И.М. Гулис**, д-р физ.-мат. наук, профессор

ОПТИЧЕСКАЯ СИСТЕМА ДИСПЕРСИОННОГО МНОГОЩЕЛЕВОГО СПЕКТРОМЕТРА

Создание приборов, позволяющих получать спектроскопическую информацию об объектах с пространственным разрешением, рассматривается в настоящее время как одно из магистральных направлений в современной спектроскопии.

Широко использующиеся в настоящее время в СПР аппаратурно-методические решения сводятся в основном к различным вариациям двух подходов: мультизональной (мультиспектральной) съемке (используются светофильтры) и гиперспектроскопии, заключающейся в последовательной съемке спектров участков изображения объекта. В обоих случаях спектрально-пространственная информация об объекте может быть сформирована в т. н. гиперкуб $I(x, y, \lambda)$ – зависимость интенсивности от двух пространственных и спектральной координат.

Целью настоящей работы является отработка решений по ключевым элементам (объективы, дисперсионное устройство) дисперсионного многощелевого спектрометра (ДМС) для видимой области спектра (400–800 нм), а также построение оптической схемы и ее оптимизация. Проектируемый ДМС должен позволять зарегистрировать гиперкуб с размерностью до $N_x \cdot N_y \cdot N_\lambda = 400 \cdot 300 \cdot 40$ в одном акте измерения. Оценки показывают, что для реализации требования по спектральному разрешению (порядка 10 нм) диаметр кружка рассеяния, даваемого коллиматорным и камерным объективами, должен не превышать 10 мкм.